6. Übung zur Vorlesung "Datenbanken" im Sommersemester 2007

– mit Musterlösungen –

Prof. Dr. Gerd Stumme, Dipl.-Inform. Christoph Schmitz

11. Juni 2007

Aufgabe 1: Rekursion

Betrachten Sie die folgende Tabelle `STRECKEN`, die die Teilstrecken und Fahrzeiten der Kasseler Straßenbahn enthält. Teilstrecken seien dabei die Abschnitte zwischen zwei benachbarten Haltestellen.

STRECKEN		
VON	NACH	ZEIT
Königsplatz	Friedrichsplatz	2
Friedrichsplatz	Rathaus	4
Rathaus	Weigelstraße	4
Weigelstraße	Universität	5
...	...	...

1. Geben Sie ein SQL-Statement an, das alle möglichen Fahrstrecken und die dazugehörigen Fahrzeiten angibt (Haltezeit sei jeweils 2 Minuten pro Haltestelle).

```
with fahrplan(von, nach, zeit) as
(
  select von, nach, zeit from strecken
  union
  select f.von, s.nach, f.zeit + s.zeit + 2
  from fahrplan f, strecken s
  where f.nach = s.von
)
select * from fahrplan
```

2. Was passiert, wenn man eine Teilstrecke Friedrichsplatz → Königsplatz einfügt?

   Wenn man Zyklen im Graphen hat, wird die oben gezeigte Anfrage nicht mehr terminieren.

3. Falls Ihr SQL-Statement in der vorigen Aufgabe ein Problem hat – können Sie dies beheben?

   Wir müssen dazu die Rekursion in der `where`-Bedingung unterbrechen, wenn ein Zyklus vorliegt:

```
with fahrplan(von, nach, zeit) as
(
  select von, nach, zeit from strecken
  union
  select f.von, s.nach, f.zeit + s.zeit + 2
  from fahrplan f, strecken s
  where f.nach = s.von and f.von != s.nach
)
select * from fahrplan
```

Aufgabe 2: Sichten

1. Formulieren Sie in SQL Sichten auf die Tabellen "Professoren", "Vorlesungen" und "Assistenten", so daß

   - keine Personalnummern zu sehen sind, sondern nur Namen
   - der Rang von Professoren nicht zu sehen ist

   Sind Ihre Sichten jeweils änderbar? (Begründung!)

```
create view profdiskret as (
  select name, raum from professoren
)

create view assidiskret(name, fachgebiet, profname) as (
  select a.name, a.fachgebiet, p.name
  from assistenten a, professoren p
  where p.persnr = a.boss
)
```

   Beide Sichten sind nicht änderbar, da jeweils kein Schlüssel für die Ursprungsrelation mehr enthalten ist.

2. Warum würde man solche Sichten wie in der vorigen Aufgabe haben wollen?

   Solche Sichten können verwendet werden, um Teile der Datenbank – also z. B. bestimmte Attribute – für eine Benutzergruppe sichtbar zu machen, ohne vollen Zugriff auf die Datenbank zu gewähren.

3. Die Tabellen “hören” und “prüfen” sollen für Studenten (lesend) zugänglich gemacht werden, und zwar so, daß jeder Student nur seine eigenen Daten sehen kann.

   Ist dies mit Hilfe von Sichten zu realisieren? Wenn ja, wie? Wenn nein: Warum nicht, und mit welchen Mitteln könnte man solche Bedingungen durchsetzen?

   Mit Sichten ist eine solche Zugangsbeschränkung auf bestimmte *Tupel* – im Gegensatz zu Attributen, s. o. – nur schwer zu realisieren. Es müßte in diesem Fall für jeden Studenten eine eigene Sicht erzeugt werden in der Art

```
-- Sicht für Student 27123
create view sicht27123 as (
   select * from prüfen
   where matrnr = 27123
)
```

   In der Regel wird man jedoch solche Zugangsbeschränkungen über entsprechende Anwendungssoftware realisieren.

Aufgabe 3: Referentielle Integrität

Geben Sie jeweils geeignete `CREATE TABLE`-Statements (Skizze genügt) mit entsprechenden Integritätsbedingen an, um folgende Bedingungen in einer Datenbank durchzusetzen:

1. Nur Lehrer aus der `LEHRER`-Tabelle können Klassenlehrer sein. Soll ein Lehrer gelöscht werden, der Klassenlehrer ist, so soll die Löschung verweigert werden.

```
create table klasse (
    ...
    klassenlehrer integer references lehrer(lehrernr)
)
```

2. Jeder Professor oder Assistenten in der Uni-Datenbank hat eine Telefonnummer. Die Telefonnummern von Professoren sind eindeutig.

   Es gibt eine Tabelle, die alle Telefonanschlüsse mit technischen Daten usw. auflistet. Wenn ein Telefonanschluss in dieser Tabelle gelöscht wird, soll der entsprechende Eintrag bei den Professoren und Assistenten auf `NULL` gesetzt werden.

```
create table telefonanschluesse (
   nummer integer primary key,
   ...
)
```

```
create table professoren (
   ...
   telefonnummer integer unique references telefonanschluss(nummer)
        on delete set null
)

create table assistenten (
   telefonnummer integer references telefonanschluss(nummer)
        on delete set null
)
```

3. Lagern Sie die Information "Wer ist der Boss von einem Assistenten" in der Uni-Datenbank in eine eigene Relation aus! Dabei soll weiterhin die 1:N-Beziehung forciert werden.

```
create table bossvon (
   assistentnr integer not null unique,
   profnr integer not null,

   primary key (assistentnr, bossnr)
)
```

Aufgabe 4: Entwurfstheorie

Betrachten Sie die funktionalen Abhängigkeiten von Kapitel 6, Folie 16:

$$F = \{A \to B, B \to A, AB \to CD, C \to D, D \to C\}$$

1. Berechnen Sie die kanonische Überdeckung!

   Hier ist nur die funktionale Abhängigkeit (FD) $AB \to CD$ interessant.

   Auf der linken Seite ist wegen $A \to B$ das Attribut B überflüssig; es bleibt also $A \to CD$ übrig. Wegen $C \to D$ und der Transitivität ist rechts D überflüssig, es bleibt also $A \to C$ übrig.

   Insgesamt haben wir also die kanonische Überdeckung

   $$F_C = \{A \to B, B \to A, A \to C, C \to D, D \to C\}$$

2. Ist diese eindeutig? Wenn ja: warum? Wenn nein: geben Sie alle kanonischen Überdeckungen an!

   Da die "kurzen" FD in der vorigen Aufgabe symmetrisch sind, können aus der FD $AB \to CD$ auch A statt B und/oder C statt D gestrichen werden. Man erhält folgende weitere mögliche kanonische Überdeckungen:

$$
\begin{aligned}
F'_C &= \{A \to B, B \to A, A \to D, C \to D, D \to C\} \\
F''_C &= \{A \to B, B \to A, B \to C, C \to D, D \to C\} \\
F'''_C &= \{A \to B, B \to A, B \to D, C \to D, D \to C\}
\end{aligned}
$$

SQL-Hausaufgabe

1. Erstellen Sie eine Sicht `notendurchschnitt(matrnr, note)`, die für jeden Studenten seinen Notendurchschnitt über alle Prüfungen ermittelt.

   Die Lösungen für die SQL-Aufgaben werden jeweils mit zwei Wochen Abstand veröffentlicht.